

Introduction to the Quantitative Analysis of Textual Data Using quanteda*

Kenneth Benoit and Paul Nulty

July 3, 2014

1 Introduction: The Rationale for quanteda

quanteda is an R package designed to simplify the process of quantitative analysis of text from start to finish, making it possible to turn texts into a structured corpus, convert this corpus into a quantitative matrix of features extracted from the texts, and to perform a variety of quantitative analyses on this matrix. The object is inference about the data contained in the texts, whether this means describing characteristics of the texts, inferring quantities of interests about the texts of their authors, or determining the tone or topics contained in the texts. The emphasis of quanteda is on *simplicity*: creating a corpus to manage texts and variables attached to these texts in a straightforward way, and providing powerful tools to extract features from this corpus that can be analyzed using quantitative techniques.

The tools for getting texts into a corpus object include:

- loading texts from directories of individual files
- loading texts “manually” by inserting them into a corpus using helper functions
- managing text encodings and conversions from source files into corpus texts
- attaching variables to each text that can be used for grouping, reorganizing a corpus, or simply recording additional information to supplement quantitative analyses with non-textual data
- recording meta-data about the sources and creation details for the corpus.

The tools for working with a corpus include:

- summarizing the corpus in terms of its language units
- reshaping the corpus into smaller units or more aggregated units
- adding to or extracting subsets of a corpus
- resampling texts of the corpus, for example for use in non-parametric bootstrapping of the texts (for an example, see Lowe and Benoit, 2013)

*This research was supported by the European Research Council grant ERC-2011-StG 283794-QUANTESS. Code contributors to the project include Alex Herzog, William Lowe, and Kohei Watanabe.

- Easy extraction and saving, as a new data frame or corpus, key words in context (KWIC)

For extracting features from a corpus, `quanteda` provides the following tools:

- extraction of word types
- extraction of word n -grams
- extraction of dictionary entries from user-defined dictionaries
- feature selection through
 - stemming
 - random selection
 - document frequency
 - word frequency
 - and a variety of options for cleaning word types, such as capitalization and rules for handling punctuation.

For analyzing the resulting *document-feature* matrix created when features are abstracted from a corpus, `quanteda` provides:

- scaling models, such as the Poisson scaling model or Wordscores
- nonparametric visualization, such as correspondence analysis
- topic models, such as LDA
- classifiers, such as Naive Bayes or k -nearest neighbour
- sentiment analysis, using dictionaries

`quanteda` is hardly unique in providing facilities for working with text – the excellent `tm` package already provides many of the features we have described. `quanteda` is designed to complement those packages, as well to simplify the implementation of the text-to-analysis workflow. `quanteda` corpus structures are simpler objects than in `tm`, as are the document-feature matrix objects from `quanteda`, compared to the sparse matrix implementation found in `tm`. However, there is no need to choose only one package, since we provide translator functions from one matrix or corpus object to the other in `quanteda`.

This vignette is designed to introduce you to `quanteda` as well as provide a tutorial overview of its features.

2 Installing `quanteda`

The code for the `quanteda` package currently resides on <http://github/kbenoit/quanteda>. From an Internet-connected computer, you can install the package directly using the `devtools` package:

```
library(devtools)
if (!require(quanteda)) install_github("quanteda", username="kbenoit")
```

This will download the package from github and install it on your computer. For other branches, for instance if you wish to install the dev branch (containing work in progress) rather than the master, you should instead run

```
install_github("quanteda", username="kbenoit", ref="dev")
```

Typically, the dev branch of a software package is under active development — so while it contains the latest updates, it is more likely to have bugs. The master branch might be missing some of the newer features, but should be more reliable.

3 Creating a corpus

3.1 Loading Documents into Quanteda

From a directory of files

The quanteda package provides several functions for loading texts from disk into a quanteda corpus.

A very common source of files for creating a corpus will be a set of text files found on a local (or remote) directory. To load in a set of these files, we will load a corpus from a set of text files using information on attributes of the text that have been conveniently stored in the text document's filename (separated by underscores). For example, for our corpus of Irish budget speeches, the filename 2010_BUDGET_03_Joan_Burton_LAB.txt tells us the year of the speech (2010), the type ("BUDGET"), a serial number (03), the first and last name of the speaker, and a party label ("LAB" for Labour).

To load this into a corpus object, we will use the corpusFromFilenames function, supplying a vector of attribute labels that correspond with the elements of the filename.

```
library(quanteda)
tmpDir <- tempdir() # create a temporary directory for example files
textfile <- "https://github.com/kbenoit/quanteda/blob/dev/texts/irishbudgets2010.zip?raw=true"
download.file(textfile, paste(tmpDir, "irishbudgets2010.zip", sep="/"),
              method="curl", extra="-L") # download this zipped archive of texts
# unzip the file to the temporary folder
unzip(paste(tmpDir, "irishbudgets2010.zip", sep="/"), exdir=tmpDir)
# list the files unzipped
list.files(paste(tmpDir, "budget_2010", sep="/"))

## [1] "2010_BUDGET_01_Brian_Lenihan_FF.txt"
## [2] "2010_BUDGET_02_Richard_Bruton_FG.txt"
## [3] "2010_BUDGET_03_Joan_Burton_LAB.txt"
## [4] "2010_BUDGET_04_Arthur_Morgan_SF.txt"
## [5] "2010_BUDGET_05_Brian_Cowen_FF.txt"
## [6] "2010_BUDGET_06_Enda_Kenny_FG.txt"
## [7] "2010_BUDGET_07_Kieran_ODonnell_FG.txt"
## [8] "2010_BUDGET_08_Eamon_Gilmore_LAB.txt"
## [9] "2010_BUDGET_09_Michael_Higgins_LAB.txt"
## [10] "2010_BUDGET_10_Ruairi_Quinn_LAB.txt"
## [11] "2010_BUDGET_11_John_Gormley_Green.txt"
## [12] "2010_BUDGET_12_Eamon_Ryan_Green.txt"
## [13] "2010_BUDGET_13_Ciaran_Cuffe_Green.txt"
## [14] "2010_BUDGET_14_Caoimhghin_OCaolain_SF.txt"
```

```
# create a corpus from the files, parsing the filenames
ieBudgets2010 <- corpusFromFileNames(paste(tmpDir, "budget_2010", sep="/"),
                                     c("year", "debate", "number", "firstname", "lastname", "party"),
                                     sep="_")
```

This creates a new quanteda corpus object where each text has been associated values for its attribute types extracted from the filename:

```
summary(ieBudgets2010)

## Corpus object contains 14 texts.
##
##           Texts Types Tokens Sentences year debate
## 2010_BUDGET_01_Brian_Lenihan_FF.txt 1655 7799      390 2010 BUDGET
## 2010_BUDGET_02_Richard_Bruton_FG.txt  956 4058      222 2010 BUDGET
## 2010_BUDGET_03_Joan_Burton_LAB.txt 1485 5770      329 2010 BUDGET
## 2010_BUDGET_04_Arthur_Morgan_SF.txt 1463 6481      349 2010 BUDGET
## 2010_BUDGET_05_Brian_Cowen_FF.txt 1473 5880      262 2010 BUDGET
## 2010_BUDGET_06_Enda_Kenny_FG.txt 1066 3875      161 2010 BUDGET
## 2010_BUDGET_07_Kieran_ODonnell_FG.txt  614 2066      141 2010 BUDGET
## 2010_BUDGET_08_Eamon_Gilmore_LAB.txt 1098 3800      208 2010 BUDGET
## 2010_BUDGET_09_Michael_Higgins_LAB.txt  447 1136       49 2010 BUDGET
## 2010_BUDGET_10_Ruairi_Quinn_LAB.txt  418 1177       60 2010 BUDGET
## 2010_BUDGET_11_John_Gormley_Green.txt  363  929       49 2010 BUDGET
## 2010_BUDGET_12_Eamon_Ryan_Green.txt  482 1513       90 2010 BUDGET
## 2010_BUDGET_13_Ciaran_Cuffe_Green.txt  423 1143       48 2010 BUDGET
## 2010_BUDGET_14_Caoimhghin_OCaolain_SF.txt 1055 3654      194 2010 BUDGET
## number  firstname lastname party
## 14 Caoimhghin OCaolain  SF
## 13  Ciaran      Cuffe    Green
## 12  Eamon      Ryan     Green
## 11  John       Gormley   Green
## 10  Ruairi     Quinn    LAB
## 09  Michael    Higgins   LAB
## 08  Eamon      Gilmore   LAB
## 07  Kieran     ODonnell  FG
## 06  Enda       Kenny     FG
## 05  Brian      Cowen     FF
## 04  Arthur     Morgan    SF
## 03  Joan       Burton    LAB
## 02  Richard    Bruton    FG
## 01  Brian      Lenihan   FF
##
## Source: /home/paul/Dropbox/code/quanteda/vignettes/* on x86_64 by paul.
## Created: Thu Jul  3 19:02:42 2014.
## Notes:  NA.
```

From a vector of texts

Another method of creating a corpus from texts is to read texts into character vectors, and then create the corpus from these. The function `getTextDir` takes a path to a directory containing some texts, and reads the texts into a character vector.

```

# load two vectors of texts (Bollinger texts from Evans et al JELS 2007)
# located in the texts directory of the github site
amicusFile <- "https://github.com/kbenoit/quanteda/blob/dev/texts/amicus_curiae.zip?raw=true"
tmpDir <- tempdir()
download.file(amicusFile, paste(tmpDir, "amicus_curiae.zip", sep="/"),
              method="curl", extra="-L") # download this zipped archive of texts
# unzip the file to the temporary folder
unzip(paste(tmpDir, "amicus_curiae.zip", sep="/"), exdir=tmpDir)

# load in the texts to a vector of texts using quanteda's getTextDir()
amicusTexts <- c(getTextDir(paste(tmpDir, "amicus/training", sep="/")),
                 getTextDir(paste(tmpDir, "amicus/testing", sep="/")))

```

In this case, the labels we are interested in (petitioner/respondent) are not clearly separated in the filename by underscores, but Now that we have the texts in a character vector, we can examine them and extract labels from the names. The code below uses the grep command, part of the standard R library, to make a list of labels from the names of the texts.

```

# change the encoding (because texts contain special symbols such as $)
amicusTexts <- iconv(amicusTexts, from="latin1", to="UTF-8")

# examine the amicusTexts object - a named character vector where the
# names of the elements are the original text filename
str(amicusTexts)

## Named chr [1:100] "In granting a strong preference in admissions to applicants from a select group of racial and e
## - attr(*, "names")= chr [1:100] "sP1P2.txt" "sR1R2.txt" "sAP01.txt" "sAP02.txt" ...

# set training class - Petitioner or Respondent, only known for the two test docs
trainclass <- factor(c("P", "R", rep(NA, length(amicusTexts)-2)))

# set test class, an attribute that could be used in classification
# here we take these from the text filenames, where
# 'AP' means Amicus brief for Petitioner
# 'AR' means Amicus brief for Respondent
testclass <- rep(NA, length(amicusTexts)) # initialize the variable
testclass[grep("AP", names(amicusTexts))] <- "AP"
testclass[grep("AR", names(amicusTexts))] <- "AR"

```

Finally, we can create a corpus from the vector of texts, and the training and testing attributes.

```

# make a corpus object with texts and training and test labels
amicusCorpus <-
  corpusCreate(amicusTexts,
               attribs = list(trainclass=trainclass, testclass=testclass),
               source = "Bollinger texts from Evans et al JELS 2007",
               notes = "Created as part of the quanteda vignette")
# summarize the first 10 texts in the corpus
summary(amicusCorpus, nmax=10)

## Corpus object contains 100 texts.
##
##      Texts Types Tokens Sentences trainclass testclass
## sP1P2.txt  2893  22907      2221         P      <NA>

```

```
## sR1R2.txt 3918 23970 1900 R <NA>
## sAP01.txt 1479 6181 435 <NA> AP
## sAP02.txt 1671 6232 644 <NA> AP
## sAP03.txt 1740 7726 696 <NA> AP
## sAP04.txt 1128 4725 431 <NA> AP
## sAP05.txt 1800 7005 583 <NA> AP
## sAP06.txt 1288 4852 381 <NA> AP
## sAP07.txt 1249 4914 330 <NA> AP
## sAP08.txt 620 1748 110 <NA> AP
##
## Source: Bollinger texts from Evans et al JELS 2007.
## Created: Thu Jul 3 19:02:46 2014.
## Notes: Created as part of the quanteda vignette.
```

Alternatively, we create a labelled corpus using the directory structure in which the files are stored. If the folder names in which the files are stored indicate values for a variable of interest.

****todo corpus from folders****

3.2 Structure of a corpus in *quanteda*

A corpus contains attributes and metadata. Metadata is information associated with the entire set of texts, such as the source or date of creation. Metadata can also be used to package supplementary material with a corpus — for example, if the corpus analysis is part of a model that includes other forms of data, they can be included here.

The attributes of a corpus are the texts themselves, and any number of other attributes which may have different values for each text.

```
names(ieBudgets2010)

## [1] "attribs" "metadata"

names(ieBudgets2010$attribs)

## [1] "texts"      "year"      "debate"    "number"    "firstname" "lastname"
## [7] "party"

ieBudgets2010$attribs$party

## [1] SF      Green Green Green LAB LAB LAB FG FG FF SF LAB
## [13] FG      FF
## Levels: SF Green LAB FG FF
```

Adding new texts

We can add new texts to a corpus using the `corpusAppend` function. For example, if we wish to add another speech to the `ieBudgets` corpus, we can use a character vector for new text, and a data frame for the attributes of that text. The column names of the new attributes dataframe should match those in the existing corpus.

```
newText <-"This is an example text to be added to the ieBudgets2010 corpus"
newAttribs <- data.frame(year="2000", debate="bud", number="99", firstname="Joe", lastname="Bloggs", party="Green" )

ieBudgets2010 <- corpusAppend(ieBudgets2010, newText, newAttribs)
```

Adding new text attributes

Alternatively, we can add a new column of attribute values without appending any new texts.

```
newAttrib <- c('gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray', 'gray')

ieBudgets2010 <- corpusAddAttributes(ieBudgets2010, newAttrib, name="suitcolor")
names(ieBudgets2010$attribs)

## [1] "texts"      "year"      "debate"    "number"    "firstname" "lastname"
## [7] "party"     "suitcolor"
```

4 Manipulating a corpus

5 Extracting Features

In order to perform statistical analysis such as document scaling, we must extract a matrix associating values for certain features with each document. In *quanteda*, we use the `dfm` function to produce such a matrix.¹

By far the most common approach is to consider each word type to be a feature, and the number of occurrences of the word type in each document the values. This is easy to see with a concrete example, so let's use the `dfm` command on the full built-in Irish budget speeches corpus. In addition to indexing into the matrix with `:`, you can also view the matrix by clicking on the `docMat` variable in the RStudio Environment pane, or using the `View()` R command.

```
docMat <- dfm(ieBudgets2010)

## Creating dfm: ... done.

docMat[1:5,1:5]
```

##		words					
## docs		€		a	abandoned	abandoning	
##	2010_BUDGET_01_Brian_Lenihan_FF.txt	75	4	143	0	0	
##	2010_BUDGET_02_Richard_Bruton_FG.txt	18	5	82	1	0	
##	2010_BUDGET_03_Joan_Burton_LAB.txt	48	11	129	1	0	
##	2010_BUDGET_04_Arthur_Morgan_SF.txt	42	7	115	0	1	
##	2010_BUDGET_05_Brian_Cowen_FF.txt	38	7	122	0	0	

¹`dfm` stands for document-feature matrix — we say ‘feature’ as opposed to ‘term’, since it is possible to use other properties of documents (e.g. ngrams or syntactic dependencies) for further analysis

6 Analyzing a document-feature matrix

By default, each row in the document-frequency matrix corresponds to a single document on disk. However, if our variable does not correspond with how the documents are stored as files, we can combine together texts and study them as if they were all part of the same document. For example, rather than counting word frequencies within each speech, we can count word frequencies as they are used by each party. We can do this using the `group` argument to `dfm`:

```
partMat <- dfm(ieBudgets2010, group="party")

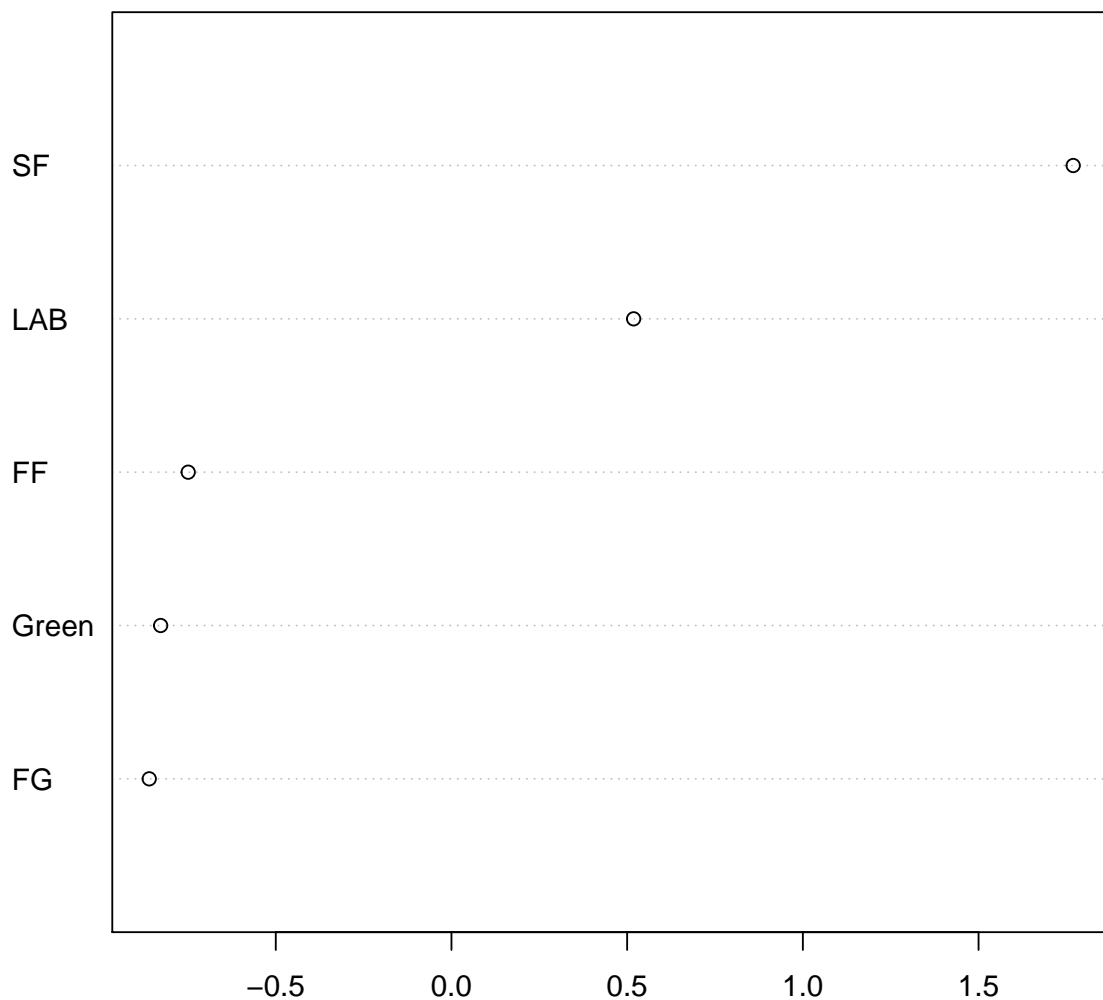
## Creating dfm: ... aggregating by group: party...complete ... done.

partMat[1:3,1:3]

##           words
## docs      €   |   a
## FF       35  5  78
## FG       31  9 162
## Green   108 23 326
```

We can now score and plot the parties using a statistical scaling technique, for example correspondence analysis (?).

```
library(ca)
model <- ca(t(partMat), nd=1)
dotchart(model$colcoord[order(model$colcoord[,1]),1], labels = model$colnames[order(model$colcoord[,1])])
```

We will discuss analysis techniques in more detail later. The `dfm` command has many optional arguments for applying standard pre-processing techniques to texts. We can choose to apply word stemming, which will sum together counts of words that have a common morphological root — for example *earn*, *earning*, *earns* and *earned* will all be counted together under the single stem *earn**. Again, this is easy to see by making a new matrix and inspecting it with `View()`

```
docMatStems <- dfm(ieBudgets2010, stem=TRUE)

## Creating dfm: ...

## Loading required package: SnowballC

## stemming ... done.
```

```
docMatStems[1:5,1:5]
```

```
##                                words
## docs                          V1  €   |   a abandon
## 2010_BUDGET_01_Brian_Lenihan_FF.txt 3 75 4 180      0
## 2010_BUDGET_02_Richard_Bruton_FG.txt 0 18 5  96      1
## 2010_BUDGET_03_Joan_Burton_LAB.txt  1 48 11 165      1
## 2010_BUDGET_04_Arthur_Morgan_SF.txt 2 42 7 144      1
## 2010_BUDGET_05_Brian_Cowen_FF.txt  3 38 7 161      0
```

Importing to and exporting from tm

The tm package uses a sparse matrix format to store document-frequency matrices. Word frequency follows a power-law distribution (Zipf’s law) — a few words are very frequent, while most words in the total corpus vocabulary don’t occur at all in the a particular document, particularly if the length of each document is a small fraction of the length of the total corpus. The result is that most cells in a document-frequency matrix have a value of zero.

In order to use less storage space, tm only stores the row and column numbers where the value is not zero. This is known as a simple triplet representation. To make use of functions from tm, we must first convert the quanteda dfm into this format.

```
tmMat<- dfm2tmformat(partMat)
```

```
## Loading required package: slam
## Loading required package: tm
## Loading required package: NLP
```

```
names(tmMat)
```

```
## [1] "i"      "j"      "v"      "nrow"   "ncol"   "dimnames"
```

References

Lowe, William and Kenneth Benoit. 2013. “Validating Estimates of Latent Traits From Textual Data Using Human Judgment as a Benchmark.” *Political Analysis* 21(3):298–313.